VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Bakalářská práce

2014 Tomáš Púček

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Dálkové ovládání robotického podvozku s využitím polohových senzorů

Remote control for robotics chassis using orientation sensors

2014 Tomáš Púček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Púček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Dálkové ovládání robotického podvozku s využitím polohových senzorů
Remote Control for Robotics Chassis Using Orientation Sensors

Zásady pro vypracování:

Navhněte aplikaci pro mobilní telefon s OS Android, pro dálkové ovládání robotického podvozku přes WiFi síť s využitím soketového rozhraní. Pro ovládání směru a rychlosti budou využity polohové senzory integorvané v MT.

1. Seznamte s fungováním a postupem měření polohových čidel používaných v mobilních telefonech s OS Android.
2. Seznamte se s programovým rozhraním OS Android pro přístup k senzorům, vytvořte testovací aplikaci.
3. Navrhněte komunikační rozhraní se zpětnou vazbou pro přenos dat z MT do řídícího počítače robotického podvozku a zpět.
4. Realizujte programové vybavení pro OS Android i pro Linux řídícího počítače. Na MT zobrazujte stav robotu.
5. Proveďte testování odezvy systému, jeho spolehlivost a stabilitu.
6. Shrňte získané zkušenosti a navrhněte možnosti dalšího rozvoje.

Seznam doporučené odborné literatury:

[1] Android SDK: http://developer.android.com/sdk/
[2] Linux Začínáme programovat, Richard Stones, Neil Metthew, COMPUTER PRESS, ISBN 80–7226–307–2

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: 7.5.2014 .....

Púček Tomáš

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Ing. Petrovi Olivkovi za odbornou pomoc a konzultaci při vytváření této práce.

Abstrakt

Obsahem této bakalářské práce je popis vývoje aplikací pro možnosti ovládání robotického podvozku, pomocí chytrým MT s operačním systémem Android. Robotický podvozek je upravený robotický vysávač iRobot Roomba a mikropočítač ALIX s Wi-Fi konektivitou. Komunikace mezi MT a podvozkem by měla probíhat přes Wi-Fi síť s využitím soketového rozhraní. Pohyb by měl být řízen pomocí akcelerometru implementovaným přímo v MT a takty by se v této aplikaci zobrazovaly data ze senzorů.

Klíčová slova

Android, IRobot Roomba, soket, akcelerometr, Google, Java, SDK, Activity

Abstract

The content of this bachelor thesis is to describe the development of applications for robotic chassis control options, using clever MT with the operating system Android. The robot chassis is a modified iRobot Roomba robotic vacuum cleaner and microcomputer ALIX with Wi-Fi connectivity. Communication between the mobile phone and the chassis should be through Wi-Fi network using a socket interface. The movement should be controlled using the accelerometer embedded in the MT and would thus appear in the application data from the sensors.

Key words

Android, IRobot Roomba, socket, accelerometer, Google, Java, SDK, Activity

Seznam použitých symbolů a zkratek

ADT	Android Development Tools
API	Application Programming Interface
DDMS	Dalvik Debug Monitor Server
DDR	Double Data Rate
GPS	Global Position Systém
IR	Infra Red
MT	mobilní telefon
OS	Operační Systém
RAM	Random Access Memory
SCI	Serial Communication Interface
SDK	Software Development Kit
Wi-Fi	Wireless Fidelity

Seznam obrázků

Obsah

1 Úvod

V poslední době se rozrostl trh s robotickými pomocníky v domácnosti. Mezi ně patří například robotický vysavač, nebo sekačka různých typů, tvarů a funkčnosti. Ovšem nic není praktičtější, než možnost ovládat takovéto zařízení, určit kam se má vydat a tam udělat svou práci. Proto není potřeba vymýšlet různé funkce, které budou mapovat svůj prostor a provádět následný pohyb zařízení. Jako ovladač může být použit jakýkoliv chytrý MT s Wi-Fi připojením. Ten už dnes používá takřka každý a na trhu existují různé modely a platformy. Nejrozšířenější platforma je dle statistik Android.

Pod pojmem Android se nachází rozsáhla platforma, zahrnující operační systém, různé nástroje a prostředky pro vývoj aplikací. Podle informací z listopadu roku 2013 měl Android 81% podíl na světových prodejích MT [1]. Z tohoto důvodu byla vytvořena aplikace komunikující s robotickým podvozkem na tuto platformu.

Aplikace by měla využívat možnosti MT a to hlavně funkce polohovacích senzorů. Většina dnes prodávaných chytrých MT obsahuje akcelerometr, ten je vhodný pro určování naklonění MT a této schopnosti budu využívat ve své aplikaci. Podle náklonu MT by měl robotický podvozek vykonávat pohyb ve stejném směru a rychlostí dle úhlu náklonu MT. Aby bylo možné komunikovat s robotickým podvozkem, musí být pro něj vytvořen program, který by zprostředkoval komunikaci pomocí soketového rozhraní přes Wi-Fi síť a řídil jeho pohyb dle přenesených dat.

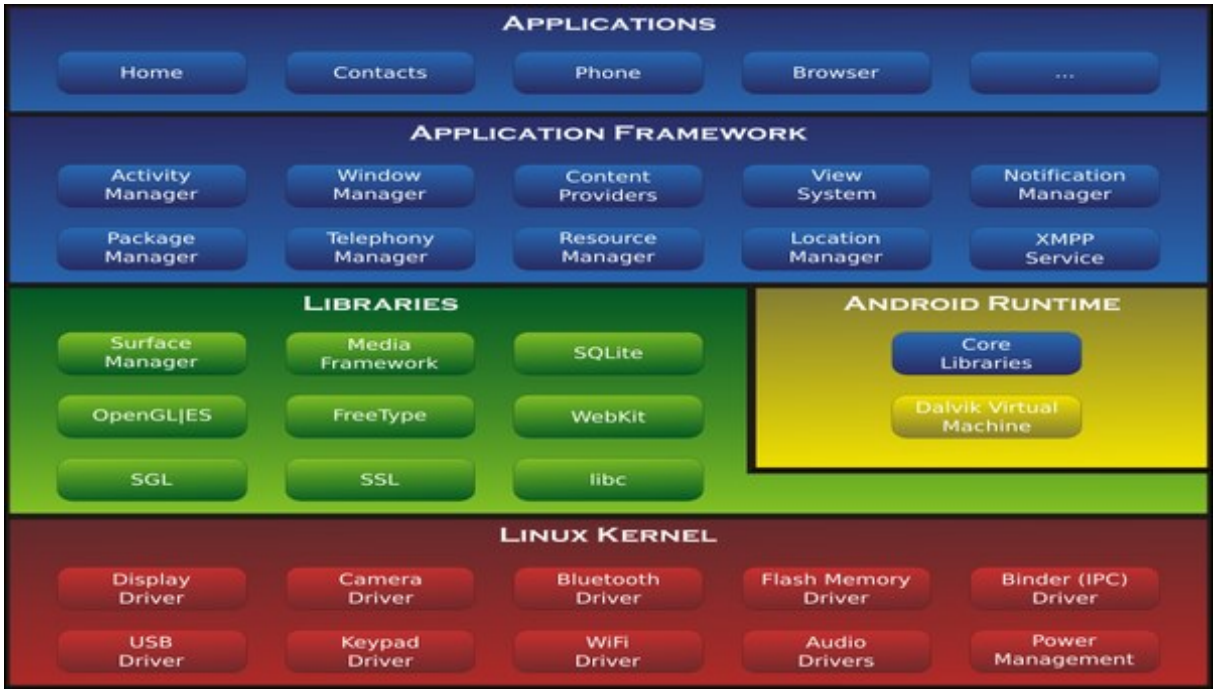
V následujících kapitolách je popsána platforma Androidu a taky vývoj aplikací jak pro tuto platformu, tak aplikaci pro robotický podvozek.

2 Platforma Android

Android je rozsáhla platforma která vznikla pod licencí open-source hlavně pro mobilní zařízení. Tato platforma v sobě zahrnuje operační systém založený na Linuxovém jádře, middleware, uživatelské rozhraní a různé aplikace.

2.1 Architektura platformy Android

Architektura systému Android je rozdělena do několika vrstev obr 1, přičemž každá vrstva využívá služeb poskytovaných vrstvou pod ní. Tím je docíleno otevřenosti systému s co největší flexibilitou a umožňující rychlý vývoj.



Obrázek 1- Architektura Android [2]

2.1.1 Popis vrstev architektury

- **Applications**: Tato vrstva obsahuje základní aplikace, které Android nabízí a aplikace třetích stran. Patří sem například program pro psaní SMS, kalendář, kontakty a další. Všechny tyto aplikace jsou napsány v jazyce Java využívající stejné knihovny.
- **Application Framework**: Obsahuje třídy a rozhraní potřebné pro vývoj aplikací. Poskytuje také abstrakci pro přístup k hardwaru, spravuje uživatelské rozhraní a aplikační zdroje. Mezi významné části patří:
  - *Views* – patří sem komponenty používané při konstrukci rozhraní aplikace. Jsou to například textová pole, tlačítka, posuvníky a jiné.
  - *Activity Manager* – část řídící životní cyklus aplikace. Aplikace musí neustále naslouchat změně svého stavu a být vždy připravena na náhle ukončení. Standardně obsahuje každá aplikace svůj vlastní proces, který vytvoří vlastní instanci virtuálního stroje Dalvik.
  - *Notification Manager*- umožňuje zobrazit upozornění v stavovém řádku.
  - *Resource Manager*- přístup k externím zdrojům jako je grafika, akcelerometr, kompas a mnohé další.
  - *Content Providers*- poskytuje abstraktní vrstvu nad jakýmikoliv daty uloženými na zařízení tak, aby byla jednoduše dostupná pro další aplikace.
- **Android Runtime**: Všechny spuštěné aplikace v OS Android běží na virtuálním stroji Dalvik. Dalvik je základní klíčová součást celého Android systému a je optimalizovaný pro běh na slabších procesorech s limitovanou operační pamětí a úsporou napájení z baterií**.** Tím je docíleno, že aplikace nemůže nijak ovlivnit chod operačního systému a také nemají přímý přístup k hardwaru MT. Tato vrstva poskytuje taktéž většinu funkcí dostupných v základních knihovnách jazyka Java.
- **Libraries**: Tato vrstva obsahuje knihovny napsané v jazyce C nebo C++ a jsou kompilované přímo pro určitý hardware telefonu. Dají se zde nalézt knihovny pro databázovou podporu SQLite, grafické knihovny nebo knihovny určené pro vykreslování oken (Surface Manager).
- **Linux Kernel**: Linuxové jádro pro správu paměti, procesů, sítí a další služby operačního systému. Toto jádro poskytuje hardwarovou abstraktní vrstvu, která dovoluje Androidu pracovat na různých platformách.

2.2 Android SDK

Je to balíček nástrojů pro tvorbu aplikací pro platformu Android, který je dostupný pro řadu operačních systému, jako jsou Windows, Mac OS nebo Linux. Nejednoduší použití tohoto balíčku je jako modul do vývojového prostředí Eclipse, který nalezneme pod zkratkou ADT [3].

Výhodou SDK je implementace nejpoužívanějších nástrojů jako jsou ddms, SDK manager a emulátor zařízení s OS Android, který umožňuje otestovat většinu funkčností aplikace bez nutnosti vlastnit toto zařízení [4]. Emulátor má i svá omezení, jako je reálné použití akcelerometru, fotoaparátu, gyroskopu, telefonování a podobné. Největší výhodou je, že si aplikaci můžeme vyzkoušet na různých

verzích systému, či různých velikostech obrazovky. Samotné programování probíhá v jazyce Java s využitím API a několik souborů XML.

Nejdůležitějším je **Manifest**, kde každá aplikace pro Android má základní soubor **AndroidManifest.xml**, uložený v hlavní složce projektu. V tomto manifestu je možno specifikovat spoustu věcí, kde mezi nejdůležitější patří [5]:

- **Oprávnění**- Aby mohla aplikace používat některé funkce systému MT, jakožto například Wi-Fi, akcelerometr, připojení k internetu, k telefonování, musí být v tomto manifestu nastavena povolení. Tato povolení jsou viditelná při samotné instalaci aplikace a uživatel jej musí potvrdit. Znemožněním použití z některých oprávnění pak zapříčiní nefunkčnost některé části aplikace nebo může způsobit pád aplikace.
  Příkladem může být povolení používat Wi-Fi spojení:

  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>

- **Minimální verze systému**- Každá verze systému Android přináší různá vylepšení oproti verzi předešlé. Některé tyto změny mají vliv na samotné SDK a přinášejí tak například nové metody, třídy a parametry nedostupné v předcházejících verzích. Nastavením minimální verzi SDK docílíme to, že aplikace nebude možné spustit v starších MT, nebo právě naopak ji bude možné spustit i na starších MT se staršími verzemi OS Android.
  Následující řádek nastaví minimální verzi 7, což odpovídá Android verze 2.1:

  <uses-sdk android:minSdkVersion="7"/>

- **Specifikace aplikace**- Jádrem manifestu je element *<application>*, kde se specifikují součásti aplikace, jako jsou aktivity *<activity>*, služby *<service>*, context providery *<provider>* a další. Dále je zde možné nastavit například jméno aplikace, ikonu a téma.
  Příkladem může být následující specifikace aplikace:

```
<application
   android:allowBackup="true"
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >
   <activity
      android:name="com.example.controller.MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
   </activity>
 </application>
```

  V tomto nastavení je specifikováno v elementu <application> možnost zálohování „allowBackup“, nastavení ikony „icon“, název aplikace „label“, téma aplikace „theme“ a že aplikace má jednu aktivitu s názvem „com.example.controller.MainActivity“ zobrazenou při spuštění.

2.2.1 Emulátor

Emulátor obr. 2 slouží pro testování aplikací bez přítomnosti fyzického zařízení. Je to virtuální stroj spuštěný v operačním systému na počítači, který se chová jako klasické zařízení s operačním systémem Android. Podle předem nastavených vlastností emulátoru, lze simulovat téměř všechna známá zařízení, na kterých v současnosti běží systém Android. Při vývoji aplikací je totiž velmi nutné vytvořit několik virtuálních zařízení, které se liší například ve verzích systému Android nebo rozlišením obrazovky. Tím se docílí, že aplikace bude fungovat a vypadat i ve skutečných zařízeních stejně.

Obrázek 2- Emulátor

2.2.2 SDK manager

Tento nástroj je důležitou součástí SDK, který slouží ke správě SDK balíčku a hlídá, aby všechny komponenty byly aktuální. S jeho pomocí můžeme doinstalovat nové, nebo starší verze SDK platforem Android nebo nové utility.

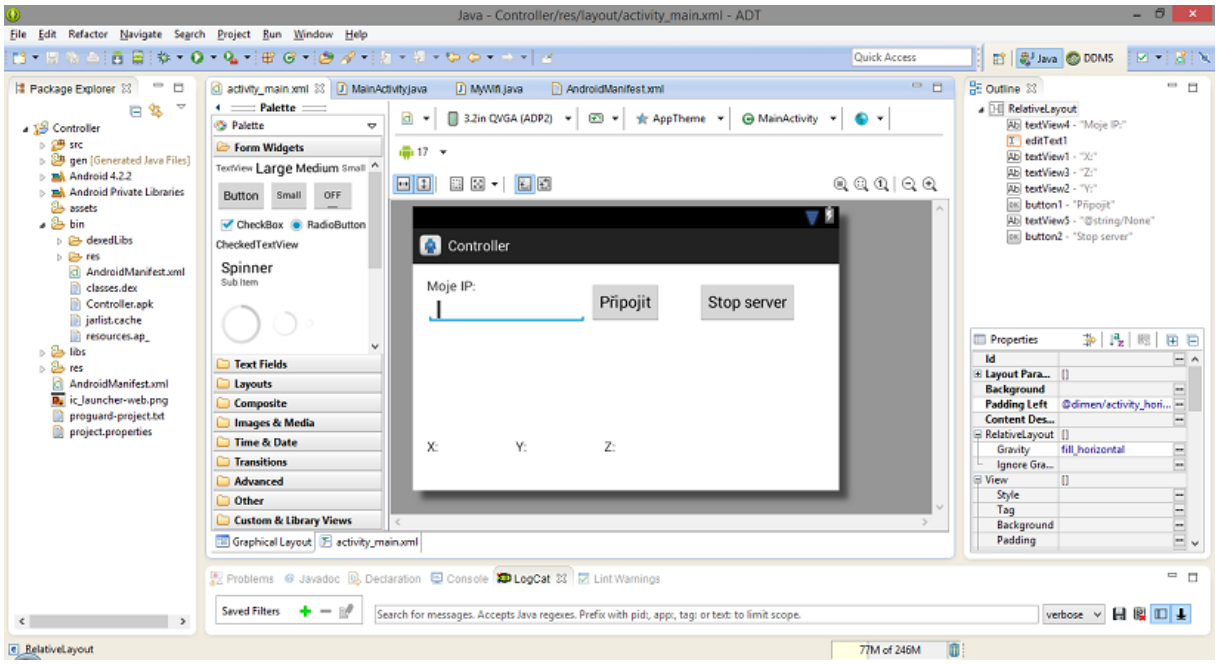
2.2.3 DDMS

DDMS slouží k ladění spuštěného programu, který běží na emulátoru, nebo na připojeném zařízení. S jeho pomocí můžeme nasimulovat různé situace, které mohou nastat v době, kdy je aplikace spuštěna na zařízení. Můžeme tak například simulovat příchozí volání, textovou zprávu, nebo odeslat na zařízení informaci o poloze zařízení. Mezi důležité vlastnosti patří zachytávání chybových hlášení, která nastanou při běhu aplikace.

2.3 Vývojové prostředí Eclipse

Nejefektivnější způsob vývoje je podle mého názoru vyvíjet aplikace pro platformu Android ve open-source vývojovém prostředí Eclipse s instalovaným pluginem ADT. Zároveň je také toto prostředí asi nejpoužívanějším při programování v jazyce Java.

Toto vývojové prostředí zahrnuje funkce, jako možnost zobrazení xml formou nákresů, zpřehledněním kódu barevným zvýrazněním, hromadné přejmenování prvků v celém projektu nebo vizuálně vkládat komponenty v grafické části projektu a přímé možnosti jejich nastavení.



Obrázek 3 – Náhled prostředí Eclipse

3 Senzory v MT s OS Android

Většina zařízení na platformě Android mají vestavěné senzory, které měří pohyb, orientaci a různé veličiny prostředí. Tyto senzory poskytují data s vysokou přesností, a jsou užitečné, pokud chceme sledovat trojrozměrný pohyb zařízení v prostoru.

Platforma Android podporuje tyto kategorie senzorů [6]:

- **Polohové senzory**- tyto senzory měří zrychlení a rotační síly podél tří os a polohu zařízení. Tato kategorie zahrnuje akcelerometry, gyroskopy, GPS, gravitační senzory a rotační vektorové senzory.
- **Senzory prostředí**- patří sem senzory, které měří různé parametry okolního prostředí, jako je teplota okolního vzduchu, tlaku, osvětlení a vlhkosti. Kategorie zahrnuje barometry, fotometry a teploměry.

Ve své aplikaci je nejvhodnější použít polohový senzor akcelerometr, který měří statické nebo dynamické zrychlení. Akcelerometr je vhodný nejen pro měření odstředivých a setrvačných sil, ale i pro určování pozice tělesa.

3.1 Použití akcelerometru v aplikaci

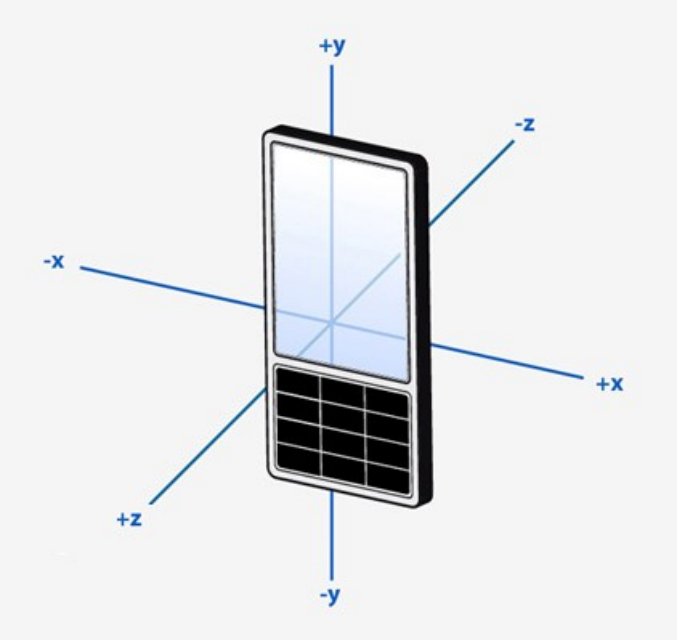
Tento senzor používaný v MT s OS Android měří hodnoty ve třech osách x, y, z obr. 4. Proto, abych mohl použít ve své aplikaci pro OS Android funkce akcelerometru, musím v hlavní třídě implementovat rozhraní *SensorEventListener*, který je deklarován v Android SDK. Toto rozhraní nám v této třídě implementuje tyto metody:

```
@Override
public void onSensorChanged(SensorEvent event) {

}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {

}
```



Obrázek 4 – Zobrazení os akcelerometru MT [11]

3.1.1 Sledování změn senzoru

První metoda „*onSensorChanged*" je volána, když nastane změna dat z nějakého ze senzorů. Druhá metoda „*onAccuracyChanged*" je volána, když dojde ke změně přesnosti senzoru. Pro detekci změny hodnot akcelerometru používám právě první metodu, kde z parametru *event* načítám hodnoty všech tří os:

```
float x = event.values[0];
float y = event.values[1];
float z = event.values[2];
```

3.1.2 Registrace posluchače změn akcelerometru

Samotná implementace tohoto rozhraní nebude volat tyto metody. V aplikaci je nutné použít třídu *SensorManager* a *Senzor*, pomocí kterých si registrujeme jako posluchače právě tuto hlavní třídu implementující rozhraní *SensorEventListener* na mnou zvolený akcelerometr.

Ukázka registrace posluchače v hlavní třídě:

```
public class MainActivity extends Activity implements SensorEventListener {

    private SensorManager sensorManager;
    private Sensor mAccelerometer;
```

```
...

@Override
protected void onCreate(Bundle savedInstanceState) {

sensorManager=(SensorManager)getSystemService(SENSOR_SERVICE);
mAccelerometer = sensorManager.
        getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sensorManager.registerListener(this, mAccelerometer,
        SensorManager.SENSOR_DELAY_NORMAL);
...
```

3.1.3 Správa posluchače změn akcelerometru

Aby bylo možné aplikaci korektně vypnout, popřípadě následně zapnout, musí aplikace spravovat tohoto posluchače. Při ukončování aplikace je zavolána metoda *onPause*, kde se musí posluchač odregistrovat a to následovně:

```
    @Override
protected void onPause() {
    sensorManager.unregisterListener(this);

    ...
```

V případě, kdy byla aplikace ukončena a uživatel ji chce znovu spustit, je volána metoda *onResume*, kde se musí posluchač opětovně registrovat:

```
    @Override
protected void onResume() {
        sensorManager.registerListener(this, mAccelerometer, SensorMa-
            nager.SENSOR_DELAY_NORMAL);

    ...
```

Tím je docíleno správné funkčnosti aplikace, jinak by mohlo dojít k jejímu pádu, který by zapříčinilo jádro systému s varovným výpisem chyby na obrazovce.

4 Robotický podvozek

Jak jsem již napsal, tak robotický podvozek se skládá z upraveného robotického vysavače iRobot Roomba 531 komunikující s mikropočítačem ALIX pomocí SCI rozhraní. Robotický vysavač je vybaven až 38 různých senzorů pro orientaci obr. 7, nabíjecí stanici a nabíjitelnou baterii o kapacitě 3000mAh. Mikropočítač je vybaven 500MHz procesorem AMD Geode s 512MB pamětí DDR RAM, na kterém je nainstalován OS Ubuntu. Proto, aby bylo možné komunikovat mezi aplikaci v MT a podvozkem, bylo nutné vytvořit malý program, který by měl funkci soketového serveru a prováděl pohyb podvozku podle přenesených dat.

4.1 Popis SCI rozhraní

Jedná se o komunikaci pomocí sériové linky RS-232 [7], mezi robotickým vysavačem a mikropočítačem, které jsou mezi sebou spojeni sériovým kabelem. Komunikace probíhá s přesně specifikovanými povely a odpověďmi určenými výrobcem vysavače. Jako hlavní zařízení (Master) v tomto případě je mikropočítač, který odesílá povely vysavači a ten je následně vykoná, popřípadě odešle odpověď [8].

4.1.1 Povely

Po přečtení knihy Hacking Roomba [8], ve které je detailní popis této komunikaci, bych povely rozdělil dle funkčnosti do čtyř kategorií:

1. **Řídící pohyb**- pomoci této kolekce příkazů, je možné ovládat pohyb vysavače ve všech směrech a taky rychlosti. Patří sem taky funkce otáčení vysavače a spouštění funkce vysávání. Například odesláním posloupnosti bytů 0x89, 0x00, 0xC8, 0x80, 0x00 je povel, který způsobí pohyb vpřed s rychlostí hodnoty 200.
2. **Načítání hodnot vestavěných senzorů**- jak sem již napsal, tak vysavač má až 38 různých senzorů, například přední senzor nárazu, rotační enkodéry, IR senzory přiblížení nebo tlačítka. Odesláním těchto dvou bytů 0x8E a 0x00 se jako odpověď odešlou data ze všech senzorů.
3. **Přehrávání tónů a melodií**- těmito příkazy je možné vyprodukovat zvuk o určitém tónu a trvání, nebo vytvářet krátké melodie.
   Odesláním posloupností bytů 0x8C, 0x0F, 0x01, 0x24, 0x40, 0x8D, 0x0F je přehrána nota C a trvání jedné sekundy.
4. **Ovládání LED diod**- tyto povely slouží ke spínání dostupných LED diod, nebo nastavit jejich intenzitu svitu.
   Pomocí těchto odeslaných bytů 0x8B, 0x20, 0x00, 0x7F je na 50 % svitu zapnutá zelená LED dioda.

Command	Opcode	Hexadecimal Values	Number of Data Bytes
START	128	0x80	0
BAUD	129	0x81	1
CONTROL	130	0x82	0
SAFE	131	0x83	0
FULL	132	0x84	0
POWER	133	0x85	0
SPOT	134	0x86	0
CLEAN	135	0x87	0
MAX	136	0x88	0
DOCK	143	0x8F	0
DRIVE	137	0x89	4
MOTORS	138	0x8A	1
LEDS	139	0x8B	3
SONG	140	0x8C	2+2N
PLAY	141	0x8D	1
SENSORS	142	0x8E	1

Obrázek 5 – Seznam příkazů s očekáváným počtem bytů [8]

4.1.2 Odpovědi

Odpovědi z robotického vysavače jsou odeslány jen v případě příchozího povelu na odeslání všech hodnot senzorů. Jedná se tedy o kolekci 26 bytů, obrázek 7 nacházející se na konci této kapitoly, ve kterých několik jednotlivých bytů obsahují hodnoty i z více senzorů. Kupříkladu první byt obsahuje informaci o stavu předních dvou nárazových senzorů.

4.2 Aplikace pro robotický podvozek

Aplikace určená pro robotický podvozek byla napsána v textovém editoru PSPad v programovacím jazyce C, nahrána do mikropočítače podvozku a tam zkompilována spolu s knižnicí [9] určenou pro sériovou komunikaci s vysavačem. K mikropočítači se připojuji pomocí klienta Putty přes Wi-Fi síť, na kterou je připojen. Použitím tohoto klienta mám dostupný linuxový terminál, ve kterém pak můžu kompilovat aplikaci pomocí příkazu:

gcc server.c roombalib.c -o server

Tato aplikace má tři důležité funkce. Tou první je, že se v ní spouští soketový server komunikující pomocí Wi-Fi sítě, na který se pak připojí MT. Na tenhle vytvořený server se odesílají data z aplikace běžící na MT s OS Android. Druhá funkce této aplikace je v navázání sériové komunikace mezi mikropočítačem a samotným vysavačem. V poslední třetí funkci spočívá dekódování příchozích dat z akcelerometru MT a vyhodnocování pohybu, kam a s jakou rychlostí se má celý podvozek vydat.

Následující kapitoly popisují tyto funkce s ukázkami kódu, jak jsou řešeny v programu určený pro robotický podvozek.

4.2.1 Soketový server

Pro funkci socketového serveru byla mnou vytvořena metoda „*CreateSocket*“, která jako IP adresu serveru používá přidělenou IP od Wi-Fi routeru a s portem 4444. Pokud je server spuštěn, pak je v terminálu vypsáno: „*Server spusten. Cekam na klienta...*“. Tím je vizuálně indikováno, jestli je server správně spuštěn a že čeká na klienta.

Výpis metody „*CreateSocket*“ z aplikace:

```
int CreateSocket()
{
   struct sockaddr_in serv_addr;
   listenfd = socket(AF_INET, SOCK_STREAM, 0);
   if (listenfd < 0)
   {
      error("ERROR creating socket");
      exit(1);
   }
   memset(&serv_addr, '0', sizeof(serv_addr));

   serv_addr.sin_family = AF_INET;
   serv_addr.sin_addr.s_addr = INADDR_ANY;
   serv_addr.sin_port = htons(4444);
   if (bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0)
   {
      error("ERROR bind");
      exit(1);
   }
   listen(listenfd, 1);
   printf("Server spusten.\nCekam na klienta...\n\n");
   return (accept(listenfd, (struct sockaddr*)NULL, NULL));
}
```

4.2.2 Sériová komunikace

Pro navázání sériové komunikace používám metodu „*roomba_init*“ která je součástí knižnice roombalib [9]. Tato metoda má jeden parametr, kam předávám řetězec znaků označující komunikační sériový port, pomocí kterého je fyzicky spojen mikropočítač a vysavač. Na tenhle port je pak zahájená komunikace s rychlostí 115200 baud. Jestliže se navázání spojení nepodaří z nějaké příčiny, pak se program ukončí.

Ukázka nastavení sériového portu voláním metody „*roomba_init*“:

```
#define DefDevice "/dev/ttyS1"
int main(int argc, char *argv[])
{
   Roomba* roomba = roomba_init( DefDevice);
   if( roomba == NULL)
   {
      printf("Chyba pripojeni... Konec.\n");
      return 0;
   }
   ...
```

Hodnota *DefDevice* určuje zmíněnou hodnotu označující port sériové komunikace.

4.2.3 Dekódování dat

Dekódování probíhá z přijatých dat odeslaných z aplikace MT. Data akcelerometru jsou odesílána ve formátu „x0.00y0.00z0.00“. Pro funkčnost robotického podvozku jsou zapotřebí pouze osy X a Y, z kterých je vybrána hodnota pro jednotlivé osy a které pak přecházejí přes podmínky. Aby bylo možné zastavit, nebo nechat stát na místě celý robotický podvozek, musel jsem vymezit hodnoty, při kterých se podvozek nebude pohybovat. Nakonec jsem došel k hodnotám X-ové osy, pro kterou platí rozmezí hodnot od 6,0 do 7,5 a Y-nové osy rozmezí od -1,4 do 1,4. Mobilní telefon je v tomto rozmezí nakloněn přibližně 45° obr. 6 a tato poloha je nulovým bodem. V případě, když jsou přijaty data akcelerometru z aplikace v MT, které odpovídají tomuto rozmezí, je odeslán povel z mikropočítače do vysavače zastavit. V opačném případě jsou data přepočítána na směr pohybu a rychlosti.

Obrázek 6 – Ukázka nulové polohy MT

Příklad dekódování pro X-ovou os:

```
int velocity = 0;
if(pos_x < 60 && pos_y > -15 && pos_y < 15 && nullP == 1)
{
   if(pos_x < 0)
      pos_x=0;
   velocity = 70-pos_x; //max 200 min 20
   velocity = velocity * 28;
   velocity = velocity / 10;
   velocity = velocity + 20; //min
   if(velocity > 250)
      velocity = 250;
   px = velocity;
   if(px != old_px)
   {
      roomba_forward_at( roomba, velocity );
      roomba_delay(CMDDELAY);
      old_px = px;
   }
}
else if(pos_x > 75 && pos_y > -15 && pos_y < 15 && nullP == 1)
{
   velocity = pos_x-74;
   velocity = velocity * 65;
   velocity = velocity / 10;
```

```
    velocity = velocity + 20; //min

    if(velocity > 250)
        velocity = 250;
    px = velocity;
    if(px != old_px)
    {
        roomba_backward_at( roomba, velocity );
        roomba_delay(CMDDELAY);
        old_px = px;
    }
}
else if(pos_x > 59 && pos_x < 76 && pos_y > -15 && pos_y < 15)
{
    if(px != 0)
    {
        roomba_stop( roomba );
        px = 0;
    }
    else if(py != 0)
    {
        roomba_stop( roomba );
        py = 0;
    }
    if(nullP != 1)
        nullP = 1;
}
...
```

Do proměnných *pos_x* a *pos_y* jsou uloženy hodnoty z akcelerometru. Proměnná s názvem *nullP* slouží k určení nulového bodu, aby se při připojení MT k soketovému serveru v případě, kdy je MT nakloněn, nezačal robotický podvozek nechtěně pohybovat. Rychlost, kterou se má podvozek pohybovat, se vypočítává a ukládá do dočasné proměnné s názvem *velocity*. Tato hodnota je pak ještě uložena do proměnné *px* a porovnávána s proměnnou *old_ px*, aby se zabránilo odesílání toho jistého povelu.

4.2.4 Ukončení komunikace

Pro ukončení komunikace mezi robotickým podvozkem a aplikací v MT, musí aplikace odeslat znak „$“. Program v mikropočítači tak pozná, že komunikace bude ukončena a že má čekat opět na dalšího klienta. Pro testování byl použit další znak „#“, který způsobí korektní ukončení programu.

Výpis kódu pro sledování těchto znaků:

```
if(sendBuff[x]== '#')
{
        printf("Konec.\n");
        roomba_stop( roomba );
        uint8_t cmd[] = {0x85};
        roomba_send( roomba , cmd, 1);
        roomba_close( roomba );
        roomba_free( roomba );
        close(connfd);
        return 0;
}
if(sendBuff[x]== '$')
{
        printf("Klient je odpojeny.\n");
        nullP = 0;
        roomba_stop( roomba );
        close(connfd);
        sleep(1);
        //connfd = CreateSocket();
        listen(listenfd, 1);
        printf("Server spusten.\nCekam na klienta...\n\n");
        connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);
}
...
```

Při ukončování programu v první podmínce výpisu kódu, je ukončena komunikace přes sériovou linku a taky zastaven server. V druhé podmínce části kódu je při ukončení komunikace mezi klientem a serverem, volána metoda *accept*, pomocí které se čeká na dalšího klienta.

byte 0	7	6	5	4	3	2	1	0
bump and wheeldrop	n/a	n/a	n/a	wheel drop caster	wheel drop left	wheel drop right	bump left	bump right

byte 1	7	6	5	4	3	2	1	0
wall	n/a	n/a	n/a	n/a	n/a	n/a	n/a	wall

byte 2	7	6	5	4	3	2	1	0
cliff left	n/a	n/a	n/a	n/a	n/a	n/a	n/a	cliff left

byte 3	7	6	5	4	3	2	1	0
cliff front left	n/a	n/a	n/a	n/a	n/a	n/a	n/a	cliff front left

byte 4	7	6	5	4	3	2	1	0
cliff front right	n/a	n/a	n/a	n/a	n/a	n/a	n/a	cliff front right

byte 5	7	6	5	4	3	2	1	0
cliff right	n/a	n/a	n/a	n/a	n/a	n/a	n/a	cliff right

byte 6	7	6	5	4	3	2	1	0
virtual wall	n/a	n/a	n/a	n/a	n/a	n/a	n/a	virtual wall

byte 7	7	6	5	4	3	2	1	0
motor over-currents	n/a	n/a	n/a	drive left	drive right	main brush	vacuum	side brush

byte 8	7	6	5	4	3	2	1	0
dirt detector left	range 0–255							

byte 9	7	6	5	4	3	2	1	0
dirt detector right	range 0–255							

byte 10	7	6	5	4	3	2	1	0
remote control	range 0–255							

byte 11	7	6	5	4	3	2	1	0
buttons	n/a	n/a	n/a	n/a	Power	Spot	Clean	Max

byte 12	7	6	5	4	3	2	1	0
distance (high byte)	range -32768 to 32767 mm							

byte 13	7	6	5	4	3	2	1	0
distance (low byte)	range -32768 to 32767 mm							

byte 14	7	6	5	4	3	2	1	0
angle (high byte)	range -32768 to 32767 mm							

byte 15	7	6	5	4	3	2	1	0
angle (low byte)	range -32768 to 32767 mm							

byte 16	7	6	5	4	3	2	1	0
charging state	values: 0:not charging, 1:charging recovery, 2:charging, 3:trickle charging, 4:waiting, 5:charging error							

byte 17	7	6	5	4	3	2	1	0
battery voltage (high byte)	range 0 to 65535 mV							

byte 18	7	6	5	4	3	2	1	0
battery voltage (low byte)	range 0 to 65535 mV							

byte 19	7	6	5	4	3	2	1	0
battery current (high byte)	range -32768 to 32767 mA							

byte 20	7	6	5	4	3	2	1	0
battery current (high byte)	range -32768 to 32767 mA							

byte 21	7	6	5	4	3	2	1	0
battery temp-erature	range -128 to 127 degrees Celsius							

byte 22	7	6	5	4	3	2	1	0
battery charge (high byte)	range 0 to 65535 mAh							

byte 23	7	6	5	4	3	2	1	0
battery charge (low byte)	range 0 to 65535 mAh							

byte 24	7	6	5	4	3	2	1	0
battery capacity (high byte)	range 0 to 65535 mAh							

byte 25	7	6	5	4	3	2	1	0
battery capacity (low byte)	range 0 to 65535 mAh							

Obrázek 7 – Seznam senzorů v pořadí přijatých bytů [8]

5 Realizace aplikace pro OS Android

Tato aplikace slouží k dálkovému řízení robotického podvozku pomocí MT s operačním systémem Android, která zprostředkovává data akcelerometru a načítá hodnoty senzorů podvozku.

Aplikace obr. 8 byla vyvíjená ve vývojovém prostředí Eclipse za pomoci modulu ADT a její funkčnost byla testována pomocí emulátoru. Pro použití na reálném zařízení, bylo nutné pomocí ADT vyexportovat aplikaci do balíčku formátu .apk, který slouží jako instalátor pro platformu Android. V dalších kapitolách je popsáno, jak tato aplikace funguje.

5.1 Struktura aplikace

Aplikace je sestavena ze tří tříd, které jsem vytvořil pro zpřehlednění zdrojového kódu a jednou z nich je hlavní třída *MainActivity.java*. Jedná se o třídu, která je rozšířena z třídy *activity*, implementující třídu *SensorEventListener.java,* jak jsem již pospal v kapitole 3.1. V této hlavní třídě jsou vytvořené instance na zbylé dvě třídy *MyWifi.java* a *ServerClient.java*, pomocí kterých jsou obsluhována z jejich názvu vyplývajících funkcí aplikace. Detailní popis těchto tříd je uveden v následujících podkapitolách.

Hned po startu aplikace je spuštěna tato hlavní třída, ve které dojde k registraci posluchače (popis v kapitole 3.1.2) změn hodnot akcelerometru. Pak dále dochází k volání metody *onSensorChanged(SensorEvent event)*, kam jsou tyto změny odeslány. V této metodě, se provede samotné načtení hodnot akcelerometru a odeslání dat serveru metodou *client.sendMessage(String.format("x%.2fy%.2fz%.2f\n", x,y,z))* v tom případě, když je MT aktuálně připojen k počítačové sítě pomocí Wi-Fi.

5.2 Hlavní třída *MainAvtivity.java*

Třída *MainActivity.java* je pro samotnou aplikaci nejdůležitější části, protože jsou tady volány všechny důležité funkce, jako například spuštění klienta pro komunikaci pomocí soketového rozhraní, nebo testování jestli je MT aktuálně připojen přes Wi-Fi k počítačové síti. K této třídě je ještě důležité podotknout, že se v ní načítá IP adresa, která se zadává do textového okénka *editText1*. Ten je nastaven tak, aby nebylo možné do něj zapisovat jiné hodnoty, které neodpovídají řetězci formátu IP adres. Pro zjednodušení zadávání IP adresy serveru, je do tohoto okénka uložená IP adresa MT, kterou je možné upravit na adresu serveru. Stlačením tlačítka „Připojit“ dojde k zavolání posluchače *wifiOnClickListener,* který je nastaven, jako funkce po stisknutí tlačítka. V této funkci, pak dojde k navázání spojení se serverem s nastavenou IP adresou (*wifiIP*) pomocí metody ***new*** *connectTask().execute("")* použité v následující ukázce kódu:

```
new connectTask().execute("");
```

```
        synchronized(this){
        try {
                wait(2000);
                if(client.is_connected){
                        bWifi.setText("Odpojit");
                }

         } catch (InterruptedException e) {
                Toast.makeText(getBaseContext(), e.getMessage(), Toast.LENGTH_LONG).show();
         }

         ...

public class connectTask extends AsyncTask<String,String,ServerClient> {

    @Override
    protected ServerClient doInBackground(String... message) {
      client = new ServerClient(wifiIP,getBaseContext(), new ServerClient.OnMessageReceived() {

                @Override
                public void messageReceived(String message) {
                        onProgressUpdate(message);
                }
       });
      client.run();
      return null;
    }

    @Override
    protected void onProgressUpdate(String... values) {
      super.onProgressUpdate(values);
      if(values[0].isEmpty())
       recaivedMessage = values.toString();
    }
}
```

Z této ukázky je vidět, že připojování probíhá asynchronně a data odesílána ze serveru jsou ukládána do proměnné *recaivedMessage*. V této proměnné se nacházejí data senzorů přijímány z podvozku. Některé nejdůležitější jako je napětí baterie, stav předních nárazových senzorů jsou pak zobrazeny na displeji MT v textovém poli *textView5* nacházející se pod editačním okénkem IP adresy. Proto, aby uživatel viděl, že došlo k otevření spojení, je tlačítko přejmenováno na „Odpojit“ (*bWifi.setText("Odpojit")*) a při jeho stisku se otevřené spojení ukončí.

5.3 Třída *MyWifi.java*

Tato třída obsahuje dvě metody k získávání údajů o připojení Wi-Fi do sítě. První metoda *MyWifiConnect()* je volána, když chci zjistit, jestli je MT právě připojený do počítačové sítě a podle toho vrací hodnotu true nebo false. Druhou metodou je *GetWifiIP()*, která slouží k získání IP adresy MT a tuto hodnotu vrací v řetězci znaků.

Výpis zdrojového kódu těchto metod:

```
public boolean MyWifiConnect(){
        ConnectivityManager connManager = (ConnectivityManager)
                _context.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo mWifi = connManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
        return mWifi.isConnected();
}

@SuppressLint("DefaultLocale") public String GetWifiIP(){
        WifiInfo info = wifi.getConnectionInfo();
        if(info == null)
                return "Chyba.";

        int ipAddress = info.getIpAddress();
        String ip = null;
        ip = String.format("%d.%d.%d.%d",
                        (ipAddress & 0xff),
                        (ipAddress >> 8 & 0xff),
                        (ipAddress >> 16 & 0xff),
                        (ipAddress >> 24 & 0xff));
        return ip;
}
```

5.4 Třída *ServerClient.java*

Třída *ServerClient.java* slouží výhradně jako funkce pro navázání a ukončení spojení se serverem pomocí soketového rozhraní. Obsahuje čtyři metody pro správu tohoto spojení:

- *IsRun()* – metoda vrací hodnotu true nebo false podle toho, jestli je navázáno spojení se serverem.
- *stopClient()* – voláním této metody je ukončené aktivní spojení se serverem.
- *sendMessage(String message)* – pomocí této metody a jejího parametru, jsou odesílána data přes aktívní spojení se serverem.
- *run()* – funkce této metody, je v navázání spojení se serverem pomocí soketového rozhraní. Dále je v jejím těle cyklus, který má na starosti načítání příchozích dat ze serveru. Cyklus je ukončen až po volání zmíněné metody *stopClient()*.

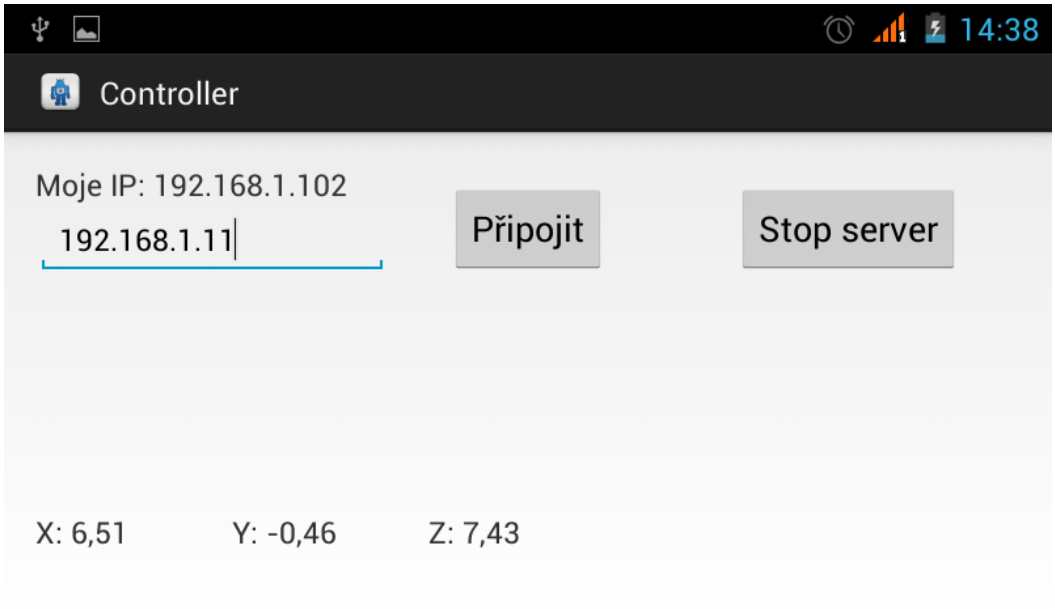
Výpis funkce *run()* ze zdrojového kódu aplikace:

```
private PrintWriter out;
private BufferedReader in;

public void run() {
    try {
        InetAddress serverAddr = InetAddress.getByName(serverIP);

        socket = new Socket(serverAddr, SERVERPORT);
        mRun = true;
        try {
            out = new PrintWriter(new BufferedWriter(new
                    OutputStreamWriter (socket.getOutputStream())), true);
            is_connected = true;
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            while (mRun) {
                serverMessage = in.readLine();
                if (serverMessage != null && mMessageListener != null) {
                    mMessageListener.messageReceived(serverMessage);
                }
                serverMessage = null;
            }
            socket.close();
        } catch (Exception e) {
            is_connected = false;
        } finally {
            socket.close();
            is_connected = false;
        }
    } catch (Exception e) {
        is_connected = false;
    }
}
```

Na začátku metody je do proměnné *InetAddress serverAddr* získána adresa serveru. Pak je vytvořen soket použitím kódu *socket = new Socket(serverAddr, SERVERPORT);*, kde *SERVERPORT* je port 4444, na který je soket otevírán. V případě, kdy je soket správně vytvořen, tak se vytvoří také streamy pro odesílání a přijímání dat. A jako poslední bych připomněl zmiňovaný cyklus ***while** (mRun)*, který stále běží, dokud proměnná *mRun* nabývá hodnoty true.



Obrázek 8 – Ukázka spuštěné testovací aplikace v MT

6 Testování systému

Testování celého systému bylo prováděno v budově školy s vlastním Wi-Fi routerem, ke kterému byl připojen robotický podvozek a MT s OS Android verze 2.1. V mikropočítači robotického podvozku byla nastavena statická IP adresa, aby nedocházelo k její změně. Tato adresa byla nastavena na hodnotu 192.168.56.11, na které bude server spuštěn a která slouží ke vzdálenému přístupu k OS Ubuntu mikropočítače. Před samotným testováním se musely nejdřív nainstalovat a spustit obě aplikace.

6.1 Spuštění programu robotického podvozku

Pro spuštění programu v robotickém podvozku, bylo nutné se k němu nejdříve připojit přes tuto počítačovou síť pomocí programu WinSCP. Pak byly tímto programem nahrány do paměti mikropočítače soubory zdrojového kódu. Tyto soubory se však musí před spuštěním zkompilovat. To se provedlo s pomocí dalšího programu s názvem Putty, přes který se tyto soubory zkompilovaly příkazem *gcc server.c roombalib.c -o server*. Po bezproblémové kompilaci, byl spuštěn výsledný program příkazem *./server*. Program po spuštění oznámil, že je server spuštěn a čeká na klienta.

6.2 Instalace aplikace pro OS Android

Jak jsem již napsal v kapitole 5, že pro spuštění aplikace na MT s Androidem, je nutné ji nejdřív nainstalovat. Při kompilaci zdrojových souborů aplikace ve vývojovém prostředí Eclipse, dojde k zabalení aplikace do instalačního souboru s příponou .apk. Tento soubor se musí zkopírovat do paměti v MT a poté nainstalovat. MT s OS Android musí být nastaven tak, aby bylo možné instalovat aplikace třetích stran. Potvrzením všech oprávnění při instalaci je aplikace nainstalována a připravena k spuštění.

6.3 Navázání komunikace

Po spuštění obou aplikací, je na řadě navázání spojení mezi nimi. Upravením IP adresy na adresu robotického podvozku v aplikaci MT obr. 8 a stisknutím na tlačítko „Připojit“ dojde ke spojení komunikace mezi robotickým podvozkem a MT. Poté jsou odesílána data akcelerometru a přijímána data z podvozku, která jsou zobrazena na aplikaci spuštěné v MT. Pro kontrolu dat jsou také v aplikaci mikropočítače zobrazovány povely, které má robotický podvozek vykonávat.

6.4 Výsledky testování

Při testování celého systému, jsem s robotickým podvozkem najezdil mnoho set metrů, ze kterého jsem zjistil několik problémů a které bylo nutné opravit. Mezi nimi patřily hlavně kolize komunikace mezi robotickým podvozkem a MT, při kterých docházelo k odesílání nekompletních dat. Po upravení komunikačního protokolu, se již tato chyba nevyskytla a odezva systému byla nad moje očekávání naprosto bezchybná. Systém má však jeden těžko řešitelný problém s akcelerometrem v MT. Během ovládání pohybu podvozku telefonem dochází k načítání a odesílaní chybných dat akcelerometru, které jsou ovlivněny malým zákmitem MT způsobený uživatelem. Zákmity způsobují cukání při pohybu podvozku. Akcelerometr má velkou citlivost a tyto zákmity se nedají jednoduše filtrovat, protože jeho úlohou je zachytávat každou takovou změnu. Když si uživatel zvykne na tuto citlivost, aby s MT příliš nekmital, pak je celý systém spolehlivý.

7 Závěr

Hlavním přínosem této bakalářské práce je kompletní návrh dvou aplikací, pracující na různých platformách, které jsou schopny spolu komunikovat pomocí Wi-Fi sítě přes soketové rozhraní. Tyto aplikace byly zkonstruovány tak, aby byl jejich chod co nejspolehlivější a pro operační systém nenáročný. Největší část vývoje zabrala aplikace pro OS Android, jelikož jsem neměl žádnou předešlou zkušenost s touto platformou. I přes to se mi podařilo splnit všechny požadované funkcionality aplikace tak, aby odpovídaly zadání. Na rozdíl od vývoje aplikace pro robotický podvozek, byla tato doba o mnoho kratší, než samotné testování celého systému, jelikož mám mnohé zkušenosti ve vývoji aplikací v programovacím jazyce C se systémem Linux.

Testováním celého systému v prostorách školy, kde jsem s robotickým podvozkem najezdil mnoho set metrů, jsem zaznamenal několik nedostatků s komunikací. Například nutnost se nacházet v blízkosti Wi-Fi routera, který příliš omezuje dosažitelnou vzdálenost pohybu podvozku. To je však řešitelné, pokud by bylo použito více těchto Wi-Fi směrovačů v počítačové síti.

Budoucí vývoj aplikace pro OS Android, bude směrován k opravám drobných nedostatků a přetvoření grafické části této aplikace, aby byla pro uživatele lépe přehledná. Pak by jistě stálo na uvážení připojení videokamery k mikropočítači robotického podvozku a úpravě jeho aplikace pro možnost odesílat data záběrů, aby bylo možné vidět na MT trasu cesty a překážky.

Literatura

[1] *Android Pushes Past 80% Market Share While Windows Phone Shipments Leap 156.0% Year Over Year in the Third Quarter, According to IDC*. In: [online]. FRAMINGHAM, Massachusetts, 12.11.2013 [Citace. 2014-04-22]. <http://www.idc.com/getdoc.jsp?containerId=prUS24108913>.

[2] Android Architecture. *elinux.org* [online]. 2011, 13. června 2011 [Citace. 2014-04-22]. Dostupné z: http://elinux.org/Android_Architecture

[3] Google Inc.: Android Developers [online]. Http://developer.android.com/, [Citace. 2014-04-22].

[4] Hashimi, S.; Komatineni, S.; MacLean, D.: *Pro Android 2*. Apress, 2010, ISBN 978-1-430-22659-8, 718 s.

[5] Murphy, M.: *Beginning Android.* Apress, 2009, ISBN 978-1-430-22419-8, 361 s.

[6] MILETTE, Greg. *Professional Android sensor programming*. Indianapolis: Wiley, c2012, xxxiii, 517 s. ISBN 978-1-118-18348-9.

[7] TIŠNOVSKÝ, Pavel. *Komunikace pomocí sériového portu RS-232C*. [online]. 4. 12. 2008 [Citace. 2014-04-24]. Dostupné z: http://www.root.cz/clanky/komunikace-pomoci-serioveho-portu-rs-232c/#ic=serial-box&icc=text-title

[8] KURT, Tod E. *Hacking Roomba* [online]. Indianapolis, IN: Wiley Pub., c2007, xviii, 436 p. [cit. 2014-04-23]. ISBN 978-047-0072-714. Dostupné z: http://www.robotiklubi.ee/_media/kursused/roomba_sumo/failid/hacking_roomba.pdf

[9] KURT, Tod E. *Roombalib -- Roomba C API*. 2006. Dostupné z: http://roombahacking.com/roombahacks/roombacmd/

[10] Hacking Roomba. *Hacking Roomba* [online]. [cit. 2014-04-28]. Dostupné z: http://hackingroomba.com/code/embedded-linux/

[11] GOVENDER, *Using the Accelerometer on Android* [online]. 3.3.2014 [cit. 2014-04-30]. Dostupné z: http://code.tutsplus.com/tutorials/using-the-accelerometer-on-android--mobile-22125

Adresářová struktura přiloženého CD

/controller	Adresář obsahuje instalátor aplikace určené pro MT s OS Android
/controller_src	Adresář obsahuje zdrojové kódy aplikace pro platformu Android
/server_src	Adresář obsahuje zdrojové kódy aplikace určené pro robotický podvozek
/doc	Adresář obsahuje dokumentaci bakalářské práce